利用資料的存取特性來提升 LINUX 檔案系統效能的設計與實作

姜美玲　兆天佑　張志維　劉光哲　林禹芃　江明勳

國立暨南國際大學資訊管理系

joanna@ncnu.edu.tw

摘要

磁碟儲存系統一直是電腦系統效能的瓶頸之一，許多研究致力於檔案系統效能的提升。由於 open source 以及 GPL 的影響，使得 Linux 蔚為風潮，許多先前檔案系統的研究成果亦相繼地被整合至 Linux 檔案系統中，而本研究希望能進一步的提升 Linux 檔案系統的效能。

由於檔案在儲存體的資料配置方式會嚴重的影響檔案系統的效能，且各檔案被存取的頻率並不相同，經常會有明顯的熱門冷門檔案的差異，特別如一些網路應用程式如 BBS，Web，…等等。因此，本研究的目的即是利用這些應用程式存在有檔案讀寫頻繁和明顯冷熱門差異之特點，將經常被存取的檔案配置在硬碟中央的 cylinders 裡，透過減少硬碟讀寫頭移動的距離來減少 seek time，以達到檔案系統效能的提升。本研究於 Linux 作業系統上實作，修改 Linux 核心，並以 BBS 為應用的實例，初步的實驗結果顯示，我們的研究能有效的提升檔案系統的效能，進而提升整體伺服器的效能。

關鍵詞：Linux、檔案系統、資料群集、BBS

Abstract

Disk-based storage systems have been the performance bottleneck of computer systems. Therefore, many research focus on improving file system performance. Since the effects of open source and GPL, Linux has gained popularity around the world. Many research results of file systems have been incorporated into Linux file systems. The goal of this study is to further improve Linux file system performance.

The way files are allocated in the storage systems would greatly affect the file system performance. In fact, file access frequencies are not all the same. Usually, files can be distinguished as hot or cold accessed files, especially for the Internet applications such as BBS, WWW, etc. Therefore, the goal of this study is to utilize the characteristics of applications having high access frequency and distinction for hot/cold accessed files, the most frequently accessed files are allocated in the center cylinders of disk. Since frequently accessed files are clustered around the center disk cylinders, disk head movement is reduced, therefore, seek time is reduced. As a result, file system performance can be improved.

We have modified Linux kernel source codes to incorporate the proposed mechanism into Linux and taken BBS as the application example. The primitive performance evaluation shows that the proposed mechanism can substantially improve file system performance, which results in improving server performance.

Keywords: Linux, File Systems, Data Clustering, BBS

一、簡介

磁碟儲存系統一直是電腦系統效能的瓶頸之一，許多研究致力於檔案系統效能的提升，如 clustering、logging、journaling、prefetching、read-ahead、write-behind 等等是常用的技術，探討的重點不外乎 data placement 以使相關的資料能儘量連續配置至儲存空間，大量資料的 I/O 方式，high availability 及 fast backup and recovery 的考量，提升 buffer cache 的 hit ratio 以減少 I/O，使用資料重整達到最佳的 data layout 等等。

而磁碟資料存取的時間包含 seek time、rotational latency、以及 transfer time[10]，其中以 seek time 佔最大部份 I/O 的時間。因此有各種 disk scheduling algorithms 相繼地被提出，將來自檔案系統的 disk requests 做重新的排序，以減少 disk seek times。除此之外，研究上都希望能將磁碟儲存空間的 data layout 最佳化，使所欲存取的資料都能儘量的連續配置，以進一步地減少存取資料時所需的 seek time 及 rotational latency。

由於檔案（file）在儲存體的資料配置方式會嚴重的影響檔案系統的效能，且各檔案或資料區塊（data block）被存取的頻率往往並不相同[9]，經常會有熱門冷門的差異，於是許多相關研究即是透過資料重整，將經常被存取的資料區塊集中儲存於磁碟中央的 blocks[1,2]，或是將資料區塊依其被存取的頻率的不同，依 organ pipe

heuristic 的配置法[14]，配置於硬碟的儲存空間，這些相關的研究[5-8,11-14]的目的都是希望透過資料區塊的配置來減少 seek time，提升檔案系統的效能。然而，系統需要 overhead 來追蹤資料區塊被存取的情形，以辨別資料區塊的冷熱門程度，而根據資料區塊的冷熱門程度來進行磁碟資料重整，以達到 organ-pipe heuristic 的 data layout 需要系統花相當大的 overhead，且磁碟重整時對系統的 response time 會有極大的影響。

這些相關的研究皆是以資料區塊（data block）而非檔案（file）被存取的頻率來區分資料是否熱門，且系統需要 overhead 來追蹤資料區塊被存取的情形，以辨別資料區塊的冷熱門程度。

然而，許多的網路應用程式如 BBS、Web、網路相簿、論壇、Blog、、、等等，存在有檔案讀寫頻繁和明顯熱門冷門的差異之特點，而資料的熱門程度是以整個檔案而非以檔案中的資料區塊為主。同時，這些應用程式本身會提供冷熱門檔案的資訊，例如：BBS 有十大熱門看版的資訊，而網路相簿經常也會有人氣指數的資訊，因此，本研究即是希望當應用程式有提供冷熱門檔案的資訊時，能夠利用這些資訊將熱門檔案配置於硬碟中央的 cylinders 裡，透過減少硬碟讀寫頭移動的距離來減少 seek time，以達到檔案系統效能的提升。

而本研究選擇以 BBS 系統為應用的實例，因其存在有檔案讀寫頻繁和明顯熱門冷門差異之特點。在 BBS 系統中的每個看板在作業系統裡都是一個目錄，看板中的文章檔案被放置在各個看板的目錄裡。由於 BBS 系統會定時統計熱門看板，且熱門看板裡的文章皆是經常被讀取的，因此可以很容易從熱門看板的目錄中得到經常被讀取的文章。

另一方面，由於 Linux [3]具有開放原始碼、穩定度高、效能高的優點，因而被廣泛應用為伺服器的作業系統，各種檔案系統相繼地被 porting 到 Linux 系統中，許多先前檔案系統的研究成果亦相繼地被整合至 Linux 檔案系統中，而本研究希望能進一步的提升 Linux 檔案系統的效能。

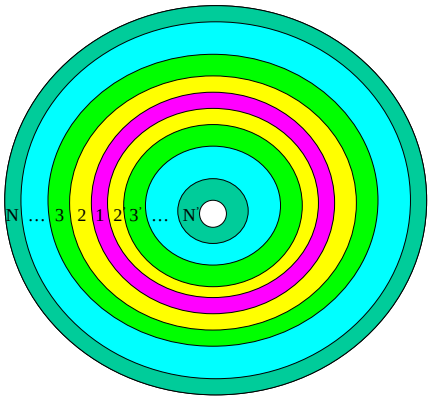
因此，本研究以 Linux 為實作平台，修改 Linux 作業系統核心，並以 BBS 為應用的實例。實驗結果顯示，我們的研究能有效的提升檔案系統的效能，進而提升整體伺服器的運作效率。

二、背景與相關研究

2.1 節描述理論上最佳化的磁碟配置，2.2 與 2.3 節描述相關的研究。2.4 節描述本研究的應用實例 - BBS 系統的資料存放方式。

2.1 最佳化的磁碟 Layout

傳統的檔案系統皆是將磁碟的儲存空間看成是一連續的空間，但研究 [14] 上指出，當資料的存取頻率是呈獨立隨機且是已知固定的分佈時，organ pipe heuristic 能達到最佳的資料配置，即資料配置在磁碟上是如圖一的配置法，將最常被存取的 blocks 放置硬碟中間 cylinder 的位置，然後根據 block 被存取的頻率依序將 blocks 放至硬碟中間 cylinder 兩側的 cylinders，以此類推，最不常用的資料會被擺在最邊緣的 cylinders。如此由於磁碟的讀取頭大部分時間都在中央移動，因此移動總距離可以有效的降低，增進檔案系統的效能。



圖一: organ pipe heuristic 的配置方式

然而，實際上資料的存取頻率並非是呈固定的分佈，也不一定是獨立隨機的，因此有 organ pipe heuristic 的變化的研究。

2.2 Adaptive Block Rearrangement

在[1,2]的研究，就是將磁碟中央的幾個 cylinders 規劃為保留區(Reserved Space)，專門放置較頻繁被讀寫的資料，使用統計資料的方式，分析最近常用的資料區塊(data clock)，在系統閒置時或使用率較低的時候，熱門的資料區塊會被 copy 至保留區內。再透過 block-remapping table 將之後的讀寫，先去找尋保留區域內的資料，以此方法減少在讀取較頻繁被讀寫的資料時的 seek time。

然而此種方法需保留一個固定的磁碟區域來備份較頻繁被讀寫的資料，且資料區塊重整的動作只能在系統的使用率較低的時候進行，不然會影響一般使用者的正常使用。

2.3 其他相關研究

在 ‘A System for Adaptive Disk Rearrangement’[13] 的研究中，最常被使用的 cylinders 會被移動到磁碟中間；’Disk Shuffling’[8] 是 Hewlett-Packard Laboratories’ Datamesh project 的一部份，它考慮 block shuffling 和 cylinder shuffling，block shuffling 和 block rearrangement

很相像，其不同在於 shuffling 簡單地交換熱門（hot）和冷門（cold）的 blocks，而 rearrangement 把熱門的資料移動到保留區域，而這也因此增加了一些儲存空間的花費。而'Smart Filesystems'[12]則是利用一個被讀寫的頻率除以檔案大小的比率，值愈大的檔案往磁碟中間移動，值愈小的檔案愈往磁碟邊邊的空間移動。

亦有研究[6]追蹤檔案的存取歷史（file access history），透過建立檔案存取圖（file access graph）的方式來分析經常一起被使用的檔案，儘量的一起被配置在相鄰的儲存空間，在離峰時段進行資料重整，然而，當檔案系統儲存很多檔案時，維護一個很大的檔案存取圖的 overhead 亦相對地變大。

2.4 BBS 系統的資料存放方式

本研究選擇以 BBS 系統為應用的實例，因其存在有檔案讀寫頻繁和明顯熱門冷門差異之特點。在 BBS 系統中的每個看板在作業系統裡都是一個目錄，看板中的文章檔案被放置在各個看板的目錄裡。由於 BBS 系統會定時統計熱門看板，且熱門看板裡的文章皆是經常被讀取的，因此可以很容易從熱門看板的目錄中得到經常被讀取的文章。

由於本研究的實作是藉由修改作業系統核心，並不更動 BBS 系統的任何程式碼，從 BBS 系統所儲存的十大熱門看板的檔案即可得知最近經常會被讀取的文章。

三、系統設計與實作

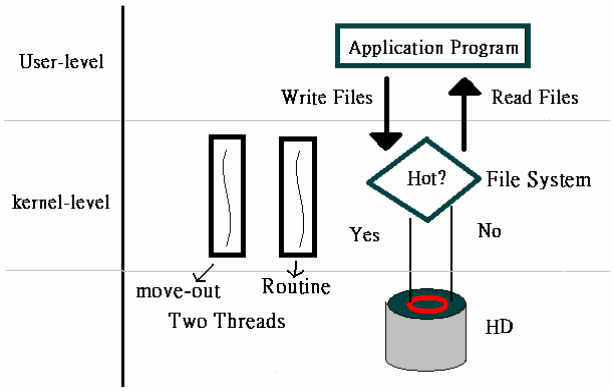
3.1 節描述本研究的系統架構，3.2 節描述系統的設計與實作。

3.1 系統架構

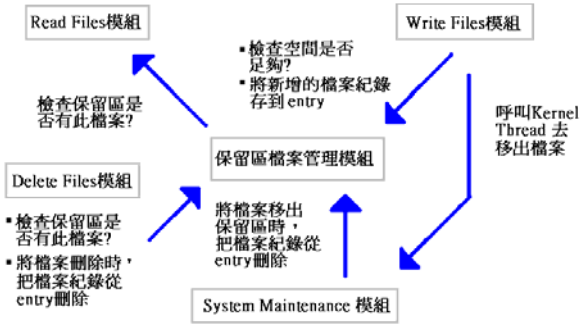
如圖二，我們在硬碟中央規劃出一塊很小的保留區，包含硬碟中央數個 cylinders，然後修改 kernel sources，使得應用程式在讀/寫檔案時，會經由 kernel 端攔截下來，判斷該檔案是否為熱門檔案，如果是的話，則至保留區讀/寫該檔案，如果不是的話，則至原本的路徑讀/寫該檔案。另外則在 kernel 產生兩個 threads 來維護保留區，一個用來在保留區快被寫滿時自動去保留區移出檔案，另一個則用來執行例行工作，例如定時載入熱門看板資料。

我們將整個系統分為以下幾個模組，分別是保留區檔案管理模組、Read Files 模組、Write Files 模組、Delete Files 模組、System Maintenance 模組，各模組之間的關係如圖三所示。

在檔案管理模組中維護了一個 table，負責記錄保留區中的檔案資訊，因此其他各模組皆要透過此一檔案管理模組來進行對保留區的各種動作，如讀檔寫檔刪檔等等。Write Files 模組負責將 User level 的應用程式寫入檔案的動作攔截，判斷是否要寫入保留區，也要將寫入保留區的檔案的資訊記錄到檔案管理模組中的 table。Read Files 模組負責將 User level 的應用程式的讀檔動作攔截，從 table 裡去比對檔案是否存在於保留區，有的話即至保留區讀取。Delete Files 模組負責將刪除檔案的動作攔截，利用 table 判斷檔案是否存在於保留區中，有的話則至保留區中將檔案刪除，同時移除在 table 中的此檔案的資訊。System Maintenance 模組負責維護保留區的空間使用和 table entry 數量，當空間即將用盡或 table entry 數量不足時，會依 LRU(Least Recently Used)的方式把檔案移出保留區且移出至有足夠空間可供使用，並且將檔案記錄從 table 中移除。



圖二：系統架構圖



圖三：模組關係圖

3.2 設計與實作

3.2.1 保留區檔案管理模組

由於系統每次存取檔案都需要做是否屬於保留區的確認動作，所以我們建立一個模組，模組由多個記錄保留區中檔案資訊的 entry 及存取

entry 的 function 構成，此 table 建立於 Memory 中，增進保留區的使用效率。

此 table 記錄保留區中存有哪些檔案，而 entry 的數量可依照保留區中 inode 的數量做調整。每個 entry 能儲存一筆檔案資訊，如圖四所示，其中，path_name 儲存該檔案的原始路徑，系統每次讀寫檔案都會比對 path_name 這一項來判斷該檔案是否屬於保留區。而每次在保留區儲存與刪除檔案時都要知道該檔案的大小以統計保留區的使用率，size 則儲存該檔案的大小，儲存單位為 Byte。而為了定時將保留區中久未被存取的檔案移出至原始路徑，我們將儲存這些檔案資訊的 entry 依存取的先後順序建成一個 link-list，並將每個 entry 的下一個和前一個 entry 的 entry#儲存於 next 與 prev 項中。

entry#	path_name	size	next	prev	co_start	co_next	co_prev
790	/IM94/abc.a	536	805	796	790	-1	-1
796	/IM95/adf.a	125	790	開始	796	-1	-1
805	/IM96/idf.a	366	結束	790	805	-1	-1

圖四：entry 示意圖

為了能夠快速搜尋到某檔案資訊儲存於哪一個 entry 中，我們使用了 Hash 的方法，將檔案資訊儲存於經 Hash function 計算所指到的 entry，使用 chaining 的方式來解決 Hash Collision 的情形。計算方式如下：

步驟一：將該檔案的原始路徑(將儲存於 path_name 中)的所有字元的 ASCII code 相加得一值 a

步驟二：entry = (a*質數) mod entry 總數，得到的數即為該筆檔案資訊欲被儲存的 entry#。

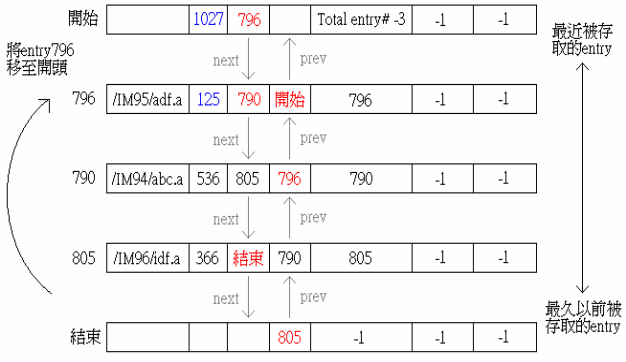
因為存取 entry 所有動作都伴隨著 kernel function sys_open( )、init( )，因此我們在 kernel 加入此模組的 function 供其他模組呼叫。

有關初始化部份，由於每個 entry 必須在系統啟動時初始化完成才可以使用，因此我們修改了 LINUX 的核心程式碼，在/init/main.c 的程式碼中加入 entry 初始化的 function：pro_init( )，此 function 使用核心函式 kmalloc( )在 physical memory 中配置適當的記憶體空間，並初始化各 entry、entry 中的各個項目及 link-list。pro_init( )會在/init/main.c 中的 start_kernel( )被呼叫以初始化所有的 entry。

有關新增檔案部份，LINUX 系統每次於保留區新增檔案時，都要將該檔案的原始路徑與檔案大小與 hash 出的數值寫進適當的 entry 中，並將該 entry 插入 link-list 中開始 entry 的下一個位置，同時更新開頭 entry 的 size 項與 co_start 項，新增檔案使用 function 為 voi pro_addfile(void)。

有關移除檔案部份，LINUX 系統每次將檔案從保留區中移除時，都要將儲存該檔案資訊的 entry 從 link-list 中移除，並將該 entry 的 path_name 項設為'\0'以表示該 entry 已未被使用，同時更新開頭 entry 的 size 項與 is_used 項，由於先前檔案資訊存入此 entry 前可能發生 collision，因此也要適當的更新 chain link，移除檔案使用的 function 為 void pro_removefile(char *path_name)。

有關讀取或覆寫檔案部份，LINUX 系統每次讀取或覆寫保留區中的檔案時，都要將儲存該檔案資訊的 entry 從 link-list 中原本的位置搬移到開頭的下一個位置，以表示該檔案最近有被存取過，如圖五。若覆寫保留區中的檔案，則 long long update_size 傳入該檔案覆寫後的大小，儲存該檔案 entry 與開頭 entry 的 size 項會被更新。若只有讀取的動作則傳入-1，function 會保留該 entry 的 size 項不做更變。讀取與修改檔案使用的 function 為 void pro_updateorder(char *update_path_name, long long update_size)。



圖五：更新 entry 於 link-list 中之次序示意圖

搜尋保留區檔案使用的 function 為 int pro_match(char *match_path_name)，若檔案存在於保留區，則此 function 會傳回整數值 1，否則傳回 0。而保留區的大小是固定的，pro_init( )初始化出來的 entry 數量是依照 inode 數量設定的，因此需要隨時掌握保留區的使用量與剩餘 entry 數以防止保留區的空間與 entry 用完。查詢保留區總使用量的 function 為 long long pro_total_used_size(void)，此 function 能傳回代表保留區總使用量的開頭 entry 的 size 項。查詢剩餘 entry 數使用 function int pro_unused_entry(void) 。當系統察覺保留區可用空間過小或可用 entry 過少時，會依照 LRU (Least Recently Used)的順序將保留區中最久未被存取的檔案移出保留區。

系統要將保留區中最久未被存取的檔案移出至原始路徑，因此系統要取得該檔案的原始路徑。此最久未被存取檔案的資訊儲存於 link-list 中結束

entry 的前一個 entry，使用 char *pro_oldest(void) function 可得到該 entry 的 path_name 項(檔案之原始路徑)。

模組在 Memory 中建立多個 entry，如果沒有適時的備份 entry，當系統關機或 crash 後就無法得知保留區中有哪些檔案，因此定時備份 entry 資訊是需要的。備份使用 void pro_backup(void)，此 function 會將模組中有被使用的 entry 資訊寫入備份檔 entry_backup.msg。重新啟動系統後模組會讀取備份檔 entry_back_up.msg，恢復所有 entry 的資訊。載入 entry 使用的 function 為 void pro_loadin(void)。

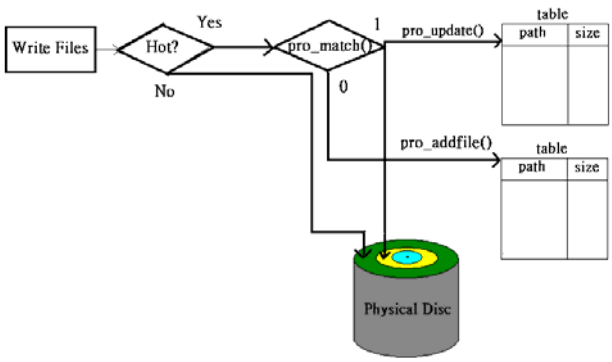
模組提供以上十個 function 給其他模組使用，這些模組可以一次使用多個本模組的 function，但為了避免多個 process 同時呼叫本模組的 function，使用本模組的 function 時要適當的使用 kernel 端的 spin_lock( )與 spin_unlock( ) function。

3.2.2 Write Files 模組

本模組攔截應用程式寫入檔案的動作， 然後區分冷門熱門，將熱門的檔案配置到保留區，冷門的則不作變更。

首先要在 kernel 底下的 sys_open( )攔截使用者寫入檔案，然後檢查是否為熱門的檔案，如果不是的話就不作任何更動，如果是熱門檔案的話，就要將檔案移至保留區，我們的作法是在完整路徑前再加上 “/reserved”，例如原本的檔案路徑是 /home/bbs/brd/11.A，改成 /reserved/home/bbs/brd/11.A 即可。

由於檢查是否為保留區的檔案必須要先判斷其是否為熱門看板的文章，而此一資訊是由應用程式所提供，會存在/home/bbs/log/day 此 log 檔中，因此必須要在開機時將此 log 檔的資料載入到記憶體以供 kernel 端的程式攔截時比對用。



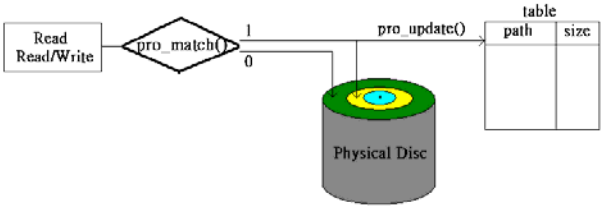
圖六：寫檔流程圖

寫入保留區同時要新增一筆記錄到檔案管理模組的 table 裡，直接使用保留區檔案管理模組中寫入 entry 的 function 即可。而如果保留區裡已經有相同的檔案時，那就直接 update 檔案管理模組裡的 table 中的記錄即可，程式流程如圖六。

3.2.3 Read Files 模組

寫入保留區同時要新增一筆記錄到檔案管理模組的 table 裡，直接使用保留區檔案管理模組中寫入 entry 的 function 即可。而如果保留區裡已經有相同的檔案時，那就直接 update 檔案管理模組裡的 table 中的記錄即可，程式流程如圖六。

在應用程式開啟檔案前，必須先檢查該檔是否存在於保留區內。我們修改 open.c 中的 sys_open( )部分，利用保留區檔案管理模組提供的 function 中的 pro_match( )來和保留區檔案管理模組中的 table 裡所存取的檔案路徑進行比對。若檔案存在於保留區中，將原始檔案路徑的開頭加上”/reserved”，如原始路徑/home/bbs/brd/ncnu/101 變更為 /reserved/home/bbs/brd/ncnu/101，系統開啟保留區中的檔案進行讀取，並呼叫保留區檔案管理模組的 pro_update( )更新檔案最後被存取的時間；若檔案不存在於保留區中，不對路徑作修改，直接進行讀取，流程圖如圖七。



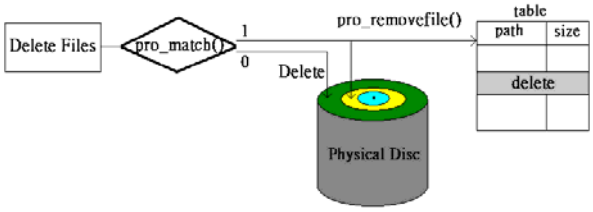
圖七：讀檔流程圖

3.2.4 Delete Files 模組

由於熱門檔案皆寫入到保留區，因此當刪除保留區的檔案時，在原始路徑下會找不到檔案可以刪除，因此必須要建立此模組以確保刪檔案動作能順利進行。

我們更改 kernel source code 的 namei.c 裡的 sys_unlink( )，在系統刪除檔案前，先檢查該檔是否存在於保留區內，比對保留區的動作藉由呼叫檔案管理模組提供的 pro_match( )來完成，如果在檔案管理模組的 table 內有該檔的記錄，則表示檔案存在於保留區中，則修改刪檔的路徑，使得保留區內的檔案可以順利刪除，若檔案記錄不存在於檔案管理模組的 table 內，則表示該檔案並沒有在保留區內，就不作任何修改路徑的動作，便可以順利的刪除該檔案。至於修改檔案路徑的作法是在原本檔案路徑之前加”/reserved” 如 /reserved/home/bbs/boards/IM94/M1234.A，便可以

順利的更改刪檔的路徑，刪完檔案之後，要呼叫檔案管理模組提供的 pro_removefile( )將該熱門檔案自檔案管理模組內的 table 中刪除，流程圖如圖八。



圖八：刪除檔案流程圖

3.2.5 System maintenance 模組

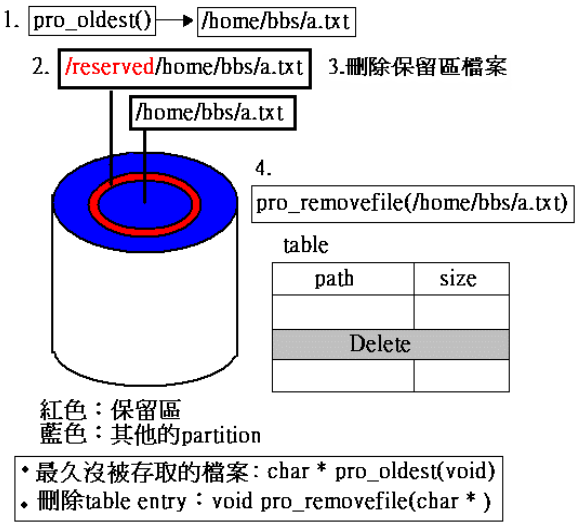
由於保留區的空間有限，再加上要寫入保留區熱門檔案的數量以及檔案大小不一定，於是必須新增一個檢查保留區剩餘空間的功能以及將檔案移出保留區的機制，來確保新增的熱門檔案能寫入保留區，並且在保留區空間不足時，能適時將檔案以 LRU (Least Recently Used)的方式移出且移出至有足夠空間可供使用。

為了確保熱門檔案寫入保留區能夠順利寫入，於是在確定為熱門檔案要寫入保留區時，必須檢查檔案管理模組的 entry 數量和保留區的空間是否足夠，我們實作一個 kernel thread，在 entry 數量不足時或保留區空間不足時，此 thread 會被 wake up 作檔案移出動作直到 entry 數量及空間足夠為止，然後此 thread 會 sleep 等候下次再被 kernel 來 wake up。

在實作時，我們在系統初始化的時候，產生一個 kernel thread(move-out)並且讓此 thread 立刻 sleep 然後放置 wait queue 等候被 kernel wake up。當使用者新增熱門看板文章時，系統會檢查 entry 數量和保留區空間是否足夠，如果足夠就將檔案寫入保留區，如果不足就會 wake up 此 move-out thread。

當 move-out thread 被 wake up 就會進行移出動作直到 entry 數量及保留區空間足夠為止，等到 entry 數量及保留區空間足夠就再將 move-out thread sleep 且放至 wait queue 等候被 kernel wake up。

至於如何將檔案由保留區移出至非保留區，我們新增一個檔案移出函式(void file_move_out(char *path))來完成這個動作，如圖九，首先，找出保留區內最久沒被存取的檔案利用 pro_oldest( )，例如：/home/bbs/a.txt。接著，在路徑前加"reserved/"並至保留區將檔案開啟，例如：/reserved/home/bbs/a.txt。再來在非保留區的 partition create 一個新檔案，例如：/home/bbs/a.txt。然後依序讀出保留區檔案內容並寫入新檔案。完成寫入後將保留區的檔案刪除。最後，利用 pro_removefile( )來移除此檔案相對的 table entry。



圖九：檔案移出方式

3.2.6 例行工作

系統每日都有三件例行工作，包含更新 BBS 每日 10 大熱門看板、定時備份 entry 資訊和定時將檔案移出。由於 BBS 系統會統計每日 10 大熱門看板，因此，系統必須每日將統計結果讀入，讓每日寫入保留區的檔案都是熱門看板的文章。而因為 entry 資訊存放在記憶體，每當重新開機或是突發狀況讓系統重新開機時，會造成 entry 資訊遺失，因此，系統必須在關機時，以及每隔一段時間就將 Table 資訊寫入硬碟作備份，來確保 entry 資訊的完整。另一方面，根據 BBS 統計結果發現每日清晨 4~5 點較少 BBS 的使用者在使用系統，此時進行檔案移出動作使用者較不易察覺，所以系統會定時在每日清晨 4~5 點檢查保留區空間和 entry 數量是否足夠供給今天的使用量，如果保留區空間和 entry 數量足夠就不進行移出動作，如果不夠就進行檔案移出動作至保留區空間和 entry 數量足夠為止。

為了完成上述三件例行工作，我們在系統初始化時產生另一個 kernel thread(routine)讓 routine thread 來完成這三件例行工作，其實作方式如下：

(1) 系統初始化時產生一個 kernel thread(routine) 。
(2) 擷取系統目前的時間。
(3) 設定讓 routine thread 每隔 30 分鐘將 entry 資訊寫入硬碟備份。
(4) 設定讓 routine thread 在每日清晨 4~5 點更新 10 大熱門看板以及檢查保留區空間和 entry 數量，在空間和 entry 數量不足時進行移出動作

直到空間和 entry 數量足夠為止。

四、實驗與效能評量

4.1 節描述本研究的實驗環境建置，包含軟硬體平台，以及測效軟體的描述，4.2 節描述我們的實驗結果，4.3 節為實驗的結論。

4.1 實驗環境建置

在實驗方面，我們所使用的硬體設備如表一，我們在 Pentium 4 1.7GHz 的機器上安裝 Fedora Core 1，並升級 Kernel 為 2.4.26 版[4]，使用 Ext2 檔案系統[3]，另外安裝 Maple BBS 作為應用範例。實驗中使用一顆 40G 的系統硬碟及一顆 4.3G 的資料硬碟，此資料硬碟是專門給 BBS 系統所使用。實驗中規劃保留區佔 2.5%的儲存空間。

表一：硬體設備

CPU	Pentium 4 1.7GHz
Memory	256MB DDR-266
System Disk	Seagate BARRACUDA ATA IV 40G (Model ST340016A), Ultra ATA-100，7200rpm，2MB cache
Data Disk	Seagate Medalist4321 4.3G (Model ST34321A), Ultra ATA，5400rpm，128KB cache

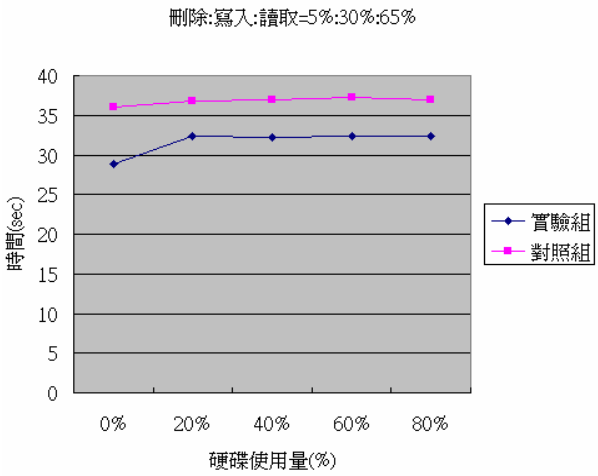
為了驗證檔案系統的修改，確實有助於整體效能的提升，我們撰寫一個 user level 的測效程式，來模擬 BBS 系統對硬碟的各種存取行為，最後記錄這些行為所花費的時間，並與原先未修改的系統加以比較，以驗證檔案系統效能的提升程度。

此一 user level 的測效程式其存取檔案的行為，分為讀取、寫入、刪除三種，依亂數而決定是何種行為，另外，讀寫與刪除的比例是可改變的控制變數，以驗證對於不同的應用環境(讀取多寫入少)，是否也有預期的效果。寫入的動作，熱門看板的文章與非熱門看板的文章之比為 7：3，保留區內有 10 個目錄，代表 BBS 的十大熱門看板，非保留區有 200 個，代表 BBS 的其他非熱門看板。測效程式亂數決定寫入的目錄後，將該檔案寫入硬碟，而檔案的大小皆設為 1 Kbytes。讀取的動作，程式會隨機開啟已寫入的檔案，將檔案內容讀進記憶體中。刪除的動作，亦是亂數決定一個已經寫入的檔案，將之刪除。另外一項測試是在硬碟中預先寫入檔案，增加亂度，來模擬硬碟長時間被讀寫後的情形。實驗總共測試十萬筆檔案的讀寫與刪除行為。實驗最終計算讀寫行為總共花費的時間，以 microsecond 為單位作為依據。相同的實驗重覆五次以求平均值。

4.2 實驗結果

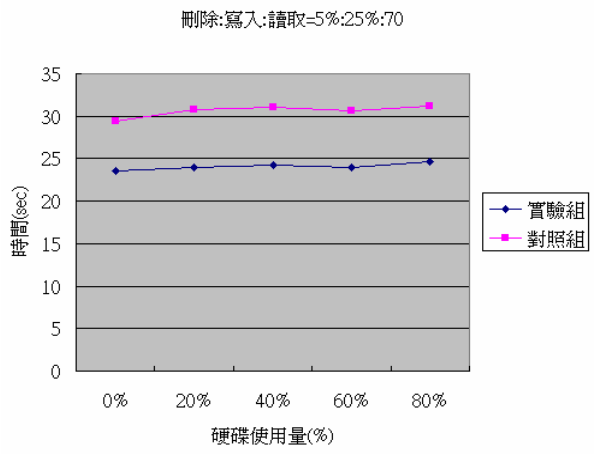
實驗中的對照組，為原先未修改的 Linux 系統，使用相同的硬碟，掛載為一塊分割區作相同的測試，以利比較。圖十至圖十三為實驗的初步測試數據，X-軸為硬碟的初始資料量，Y-軸為測效程式執行所花的時間。

圖十顯示當檔案的刪除、寫入、讀取的比例是 5%、30%、65%時，實驗組在硬碟使用量為 0%時，相對於對照組，已有 20%的效能提升。在硬碟 80%被使用時也有 12%的效果，效能提升十分顯著。



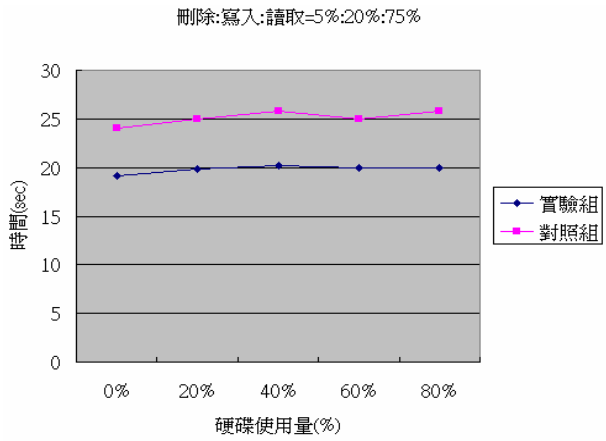
圖十：實驗結果(5%刪除,30%寫入,65%讀取)

圖十一顯示當檔案的刪除、寫入、讀取的比例是 5%、25%、70%時，實驗組的效能相對於對照組皆達到 20%左右的效能提升。由此可推測隨著讀取比例的上升和寫入比例的下降，本系統可以為檔案系統帶來實質的幫助。



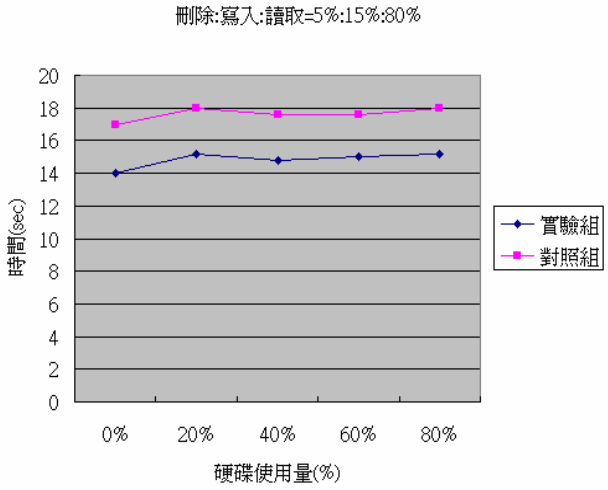
圖十一：實驗結果(5%刪除,25%寫入,70%讀取)

圖十二顯示當檔案的刪除、寫入、讀取的比例是 5%、20%、75%時，硬碟經過長時間的存取後，若不經重整資料會散佈於各個磁區，導致 seek time 增加，系統整體效能下降。比較實驗組和對照組，在硬碟使用比例增加時的時間變化，可以看出實驗組的變化較小，表示在經過長時間的使用後，仍能保持良好效能。



圖十二：實驗結果(5%刪除,20%寫入,75%讀取)

圖十三顯示當檔案的刪除、寫入、讀取的比例是 5%、15%、80%時，隨著硬碟使用量百分比的提高，效能的提升依然顯著。



圖十三：實驗結果(5%刪除,15%寫入,80%讀取)

4.3 實驗結論

從圖十至圖十三的實驗結果可以得知，在大量寫入的環境下，由於保留區的大小以及保留區管理模組的 entry 數有限，容易造成移出的動作頻繁而影響整體效能，但整體效能仍有提升。但當讀取和寫入的比例越是懸殊，即系統中越多的讀取行為，會使整體效能的提升會越顯著。以長期的使用來看，當資料在硬碟的散佈趨於混亂的情況下，當硬碟的使用量越滿時，效能的提升依然顯著。實驗中效能的提升達 12-22%。

五、結論與未來工作

本研究利用許多應用程式存在有檔案讀寫頻繁和明顯冷熱門差異之特點，將經常被存取的檔案配置在硬碟中央的 cylinders 裡，透過減少硬碟讀寫頭移動的距離來達到檔案系統效能的提升。本研究實作於 Linux 系統上，並以 BBS 系統為應用實例，藉由修改 Linux 作業系統核心，並不需更動 BBS 系統的程式碼。

由實驗數據顯示本系統很適合用於網路相關的軟體或系統上，Linux 本身提供了優良的 server 平台，所以像是以一般網站、論壇、網路相簿、BBS(電子佈告欄) 等應用程式存在有檔案讀寫頻繁和明顯冷熱門差異之特點，都是很好的應用，可以使 server 的負擔減輕，增加效能。再進一步的，由實驗數據中可以發現，經過保留區的設計，只需要極少的硬碟空間，就能使得整個檔案系統獲得提升，也就是能有效的提升整體 server 的效能，因此非常適用於以 Linux 為平台的 server 以及此平台下的網路應用程式。

但是由於寫入檔案的方式依照每種軟體或系統不盡相同，因此在攔截寫入檔案時必須要研究每套軟體或系統的寫檔部份來做調整，例如有些應用程式並非使用 open system call 做寫入的動作，而是先將檔案寫成暫存檔，再利用 rename()或者 hard link 的方式去產生真正的檔案，針對這些軟體，就不能單純的只攔截 fopen()或者 open()，而必須再去攔截 rename 的動作，來區分檔案是熱門或冷門的。

因此在未來工作方面，一方面要將本系統設計能相容更多的檔案寫入方式以因應各式各樣的網路應用程式，另一方面由於目前效能測試的方式是模擬網路應用程式的運作模式來測試檔案系統效能的提升程度，未來將對實際的網路應用程式如 BBS 做長期的效能監測，或是記錄 BBS 實際的讀寫行為來做測效，才能更正確地反應效能提升的程度。

參考文獻

[1] S. Akyurek and K. Salem, “Adaptive Block Rearrangement,” ACM Transactions on Computer Systems, 13, (2), 89-121, 1995.

[2] S. Akyurek and K. Salem, “Adaptive Block Rearrangement Under UNIX,”

Software-Practice and Experience, 27, (1), pp.1-23, January 1997.

[3] Daniel P. Bovet and Marco Cesati, *Understanding the LINUX Kernel*, 2nd edition, O'Reilly, December 2002.

[4] Cross-Referencing Linux, http://lxr.LINUX.no/source/.

[5] D. D. Grossman and H. F. Silverman, "Placement of records on a secondary storage device to minimize access time," Journal of ACM, vol. 20, no.3 (July), pp. 429-438, 1973.

[6] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, and T. E. Anderson, "Improving the Performance of Log-Structured File Systems with Adaptive Methods," Proceedings of the Sixteenth ACM Symposium on Operating System Principles, Saint Malo, France, October 5-8, 1997.

[7] M. McDonald and R. Bunt, "Improving File System Performance by Dynamically Restructuring Disk Space," Phoenix Conference on Computers and Communication, pp. 264-269, Mar. 1989.

[8] C. Ruemmler and J. Wilkes, "Disk Shuffling," Technical Report HPL-91-156, Hewlett-Packard Laboratories, Palo Alto, CA, (October 28, 1991).

[9] C. Ruemmler and J. Wilkes, "UNIX Disk Access Patterns," Proceedings of the 1993 Winter USENIX, San Diego, CA, Jan. 1993.

[10] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 6th edition, John Wiley & Sons, INC., 2002

[11] C. Staelin and H. Garcia-Molina, "Clustering Active Disk Data to Improve Disk Performance," Technical Report CS-TR-283-90, Department of Computer Science, Princeton University, (September 1990).

[12] C. Staelin and H. Garcia-Molina, "Smart Filesystems," Proceedings of the 1991 Winter USENIX, pp. 45-51, Dallas, TX, (1991).

[13] P. Vongsathorn and S. D. Carson, "A System for Adaptive Disk Rearrangement," Software-Practice and Experience, 20, (3), 225-242, (1990).

[14] C. K. Wong, "Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems", ACM Comput. Surv. 12, 2, pp. 167-178, 1980.